Annex 0 (informative)

Grammar summary [gram]

1 This summary of C++ syntax is intended to be an aid to comprehension. It is not an exact statement of the language. In particular, the grammar described here accepts a superset of valid C++ constructs. Disambiguation rules (_stmt.ambig_, _dcl.spec_, _class.member.lookup_) must be applied to distinguish expressions from declarations. Further, access control, ambiguity, and type rules must be used to weed out syntactically valid but meaningless constructs.

1.1 Keywords **[gram.key]**

1 New context-dependent keywords are introduced into a program by typedef (_dcl.typedef_), namespace (_namespace.def_), class (_class_), enumeration (_dcl.enum_), and template (_temp_) declarations.

```
        typedef-name:
                identifier
        namespace-name:
                original-namespace-name
                namespace-alias

        original-namespace-name:
                identifier

        namespace-alias:
                identifier
        class-name:
                identifier
                template-id
        enum-name:
                identifier
        template-name:
                identifier
```

Note that a *typedef-name* naming a class is also a *class-name* (_class.name_).

1.2 Lexical conventions **[gram.lex]**

```
        hex-quad:
                hexadecimal-digit hexadecimal-digit hexadecimal-digit hexadecimal-digit

        universal-character-name:
                \u hex-quad
                \U hex-quad hex-quad

        preprocessing-token:
                header-name
                identifier
                pp-number
                character-literal
                string-literal
                preprocessing-op-or-punc
                each non-white-space character that cannot be one of the above
        token:
                identifier
                keyword
                literal
                operator
                punctuator
        header-name:
                <h-char-sequence>
                "q-char-sequence"
        h-char-sequence:
                h-char
                h-char-sequence h-char
        h-char:
                any member of the source character set except
                        new-line and >
        q-char-sequence:
                q-char
                q-char-sequence q-char
        q-char:
```

```
        any member of the source character set except
                new-line and "
pp-number:
        digit
        . digit
        pp-number digit
        pp-number nondigit
        pp-number e sign
        pp-number E sign
        pp-number .
identifier:
        nondigit
        identifier nondigit
        identifier digit
nondigit: one of
        universal-character-name
        _ a b c d e f g h i j k l m
          n o p q r s t u v w x y z
          A B C D E F G H I J K L M
          N O P Q R S T U V W X Y Z
digit: one of
        0 1 2 3 4 5 6 7 8 9

preprocessing-op-or-punc: one of
{       }       [       ]       #       ##      (       )
<:      :>      <%      %>      %:      %:%:    ;       :       ...
new     delete  ?       ::      .       .*
+       -       *       /       %       ^       &       |       ~
!       =       <       >       +=      -=      *=      /=      %=
^=      &=      |=      <<      >>      >>=     <<=     ==      !=
<=      >=      &&      ||      ++      --      ,       ->*     ->
and     and_eq  bitand  bitor   compl   not     not_eq  or      or_eq
xor     xor_eq

literal:
        integer-literal
        character-literal
        floating-literal
        string-literal
        boolean-literal
integer-literal:
        decimal-literal integer-suffixopt
        octal-literal integer-suffixopt
        hexadecimal-literal integer-suffixopt
decimal-literal:
        nonzero-digit
        decimal-literal digit
octal-literal:
        0
        octal-literal octal-digit
hexadecimal-literal:
        0x hexadecimal-digit
        0X hexadecimal-digit
        hexadecimal-literal hexadecimal-digit
nonzero-digit: one of
        1  2  3  4  5  6  7  8  9
octal-digit: one of
        0  1  2  3  4  5  6  7
hexadecimal-digit: one of
        0  1  2  3  4  5  6  7  8  9
        a  b  c  d  e  f
        A  B  C  D  E  F
integer-suffix:
        unsigned-suffix long-suffixopt
        long-suffix unsigned-suffixopt
unsigned-suffix: one of
        u  U
long-suffix: one of
        l  L
character-literal:
        'c-char-sequence'
        L'c-char-sequence'
c-char-sequence:
        c-char
        c-char-sequence c-char

c-char:
```

```
                any member of the source character set except
                        the single-quote ', backslash \, or new-line character
                escape-sequence
                universal-character-name
        escape-sequence:
                simple-escape-sequence
                octal-escape-sequence
                hexadecimal-escape-sequence
        simple-escape-sequence: one of
                \'  \"  \?  \\
                \a  \b  \f  \n  \r  \t  \v
        octal-escape-sequence:
                \ octal-digit
                \ octal-digit octal-digit
                \ octal-digit octal-digit octal-digit
        hexadecimal-escape-sequence:
                \x hexadecimal-digit
                hexadecimal-escape-sequence hexadecimal-digit
        floating-literal:
                fractional-constant exponent-partopt floating-suffixopt
                digit-sequence exponent-part floating-suffixopt
        fractional-constant:
                digit-sequenceopt . digit-sequence
                digit-sequence .
        exponent-part:
                e signopt digit-sequence
                E signopt digit-sequence
        sign: one of
                +  -
        digit-sequence:
                digit
                digit-sequence digit
        floating-suffix: one of
                f  l  F  L
        string-literal:
                "s-char-sequenceopt"
                L"s-char-sequenceopt"
        s-char-sequence:
                s-char
                s-char-sequence s-char
        s-char:
                any member of the source character set except
                        the double-quote ", backslash \, or new-line character
                escape-sequence
                universal-character-name
        boolean-literal:
                false
                true
```

1.3 *Basic concepts* **[gram.basic]**

```
        translation-unit:
                declaration-seqopt
```

1.4 *Expressions* **[gram.expr]**

```
        primary-expression:
                literal
                this
                :: identifier
                :: operator-function-id
                :: qualified-id
                ( expression )
                id-expression

        id-expression:
                unqualified-id
                qualified-id
        id-expression:
                unqualified-id
                qualified-id

        unqualified-id:
                identifier
                operator-function-id
                conversion-function-id
                ~ class-name
                template-id
        qualified-id:
```

```
        nested-name-specifier templateopt unqualified-id
nested-name-specifier:
        class-or-namespace-name :: nested-name-specifieropt

class-or-namespace-name:
        class-name
        namespace-name
postfix-expression:
        primary-expression
        postfix-expression [ expression ]
        postfix-expression ( expression-listopt )
        simple-type-specifier ( expression-listopt )
        postfix-expression . templateopt ::opt id-expression
        postfix-expression -> templateopt ::opt id-expression
        postfix-expression . pseudo-destructor-name
        postfix-expression -> pseudo-destructor-name
        postfix-expression ++
        postfix-expression --
        dynamic_cast < type-id > ( expression )
        static_cast < type-id > ( expression )
        reinterpret_cast < type-id > ( expression )
        const_cast < type-id > ( expression )
        typeid ( expression )
        typeid ( type-id )

expression-list:
        assignment-expression
        expression-list , assignment-expression
pseudo-destructor-name:
        ::opt nested-name-specifieropt type-name :: ~ type-name
        ::opt nested-name-specifieropt ~ type-name
unary-expression:
        postfix-expression
        ++  cast-expression
        --  cast-expression
        unary-operator cast-expression
        sizeof unary-expression
        sizeof ( type-id )
        new-expression
        delete-expression
unary-operator: one of
        *  &  +  -  !  ~
new-expression:
        ::opt new new-placementopt new-type-id new-initializeropt
        ::opt new new-placementopt ( type-id ) new-initializeropt
new-placement:
        ( expression-list )
new-type-id:
        type-specifier-seq new-declaratoropt
new-declarator:
        ptr-operator new-declaratoropt
        direct-new-declarator
direct-new-declarator:
        [ expression ]
        direct-new-declarator [ constant-expression ]
new-initializer:
        ( expression-listopt )
delete-expression:
        ::opt delete cast-expression
        ::opt delete [ ] cast-expression
cast-expression:
        unary-expression
        ( type-id ) cast-expression
pm-expression:
        cast-expression
        pm-expression .* cast-expression
        pm-expression ->* cast-expression
multiplicative-expression:
        pm-expression
        multiplicative-expression * pm-expression
        multiplicative-expression / pm-expression
        multiplicative-expression % pm-expression
additive-expression:
        multiplicative-expression
        additive-expression + multiplicative-expression
        additive-expression - multiplicative-expression
```

```
shift-expression:
        additive-expression
        shift-expression << additive-expression
        shift-expression >> additive-expression
relational-expression:
        shift-expression
        relational-expression < shift-expression
        relational-expression > shift-expression
        relational-expression <= shift-expression
        relational-expression >= shift-expression
equality-expression:
        relational-expression
        equality-expression == relational-expression
        equality-expression != relational-expression
and-expression:
        equality-expression
        and-expression & equality-expression
exclusive-or-expression:
        and-expression
        exclusive-or-expression ^ and-expression
inclusive-or-expression:
        exclusive-or-expression
        inclusive-or-expression | exclusive-or-expression
logical-and-expression:
        inclusive-or-expression
        logical-and-expression && inclusive-or-expression
logical-or-expression:
        logical-and-expression
        logical-or-expression || logical-and-expression
conditional-expression:
        logical-or-expression
        logical-or-expression ? expression : assignment-expression
assignment-expression:
        conditional-expression
        logical-or-expression assignment-operator assignment-expression
        throw-expression
assignment-operator: one of
        =  *=  /=  %=   +=  -=  >>=  <<=  &=  ^=  |=
expression:
        assignment-expression
        expression , assignment-expression
constant-expression:
        conditional-expression
```

1.5 Statements **[gram.stmt.stmt]**

```
statement:
        labeled-statement
        expression-statement
        compound-statement
        selection-statement
        iteration-statement
        jump-statement
        declaration-statement
        try-block

labeled-statement:
        identifier : statement
        case constant-expression : statement
        default : statement
expression-statement:
        expressionopt ;
compound-statement:
         { statement-seqopt }
statement-seq:
        statement
        statement-seq statement
selection-statement:
        if ( condition ) statement
        if ( condition ) statement else statement
        switch ( condition ) statement
condition:
        expression
        type-specifier-seq declarator = assignment-expression
iteration-statement:
        while ( condition ) statement
        do statement  while ( expression ) ;
        for ( for-init-statement conditionopt ; expressionopt ) statement
for-init-statement:
        expression-statement
        simple-declaration
jump-statement:
        break ;
        continue ;
        return expressionopt ;
        goto identifier ;
declaration-statement:
        block-declaration
```

1.6 Declarations **[gram.dcl.dcl]**

```
        declaration-seq:
                declaration
                declaration-seq declaration
        declaration:
                block-declaration
                function-definition
                template-declaration
                explicit-instantiation
                explicit-specialization
                linkage-specification
                namespace-definition
        block-declaration:
                simple-declaration
                asm-definition
                namespace-alias-definition
                using-declaration
                using-directive
        simple-declaration:
                decl-specifier-seqopt init-declarator-listopt ;

        decl-specifier:
                storage-class-specifier
                type-specifier
                function-specifier
                friend
                typedef
        decl-specifier-seq:
                decl-specifier-seqopt decl-specifier
        storage-class-specifier:
                auto
                register
                static
                extern
                mutable
        function-specifier:
                inline
                virtual
                explicit
        typedef-name:
                identifier
        type-specifier:
                simple-type-specifier
                class-specifier
                enum-specifier
                elaborated-type-specifier
                cv-qualifier
        simple-type-specifier:
                ::opt nested-name-specifieropt type-name
                char
                wchar_t
                bool
                short
                int
                long
                signed
                unsigned
                float
                double
                void
        type-name:
                class-name
                enum-name
                typedef-name
        elaborated-type-specifier:
                class-key ::opt nested-name-specifieropt identifier
                enum ::opt nested-name-specifieropt identifier
                typename ::opt nested-name-specifier identifier
                typename ::opt nested-name-specifier identifier < template-argument-list >
        enum-name:
                identifier
        enum-specifier:
                enum identifieropt { enumerator-listopt }

        enumerator-list:
                enumerator-definition
                enumerator-list , enumerator-definition
        enumerator-definition:
                enumerator
                enumerator = constant-expression
        enumerator:
                identifier
        namespace-name:
                original-namespace-name
                namespace-alias
        original-namespace-name:
                identifier

        namespace-definition:
                named-namespace-definition
                unnamed-namespace-definition

        named-namespace-definition:
                original-namespace-definition
                extension-namespace-definition

        original-namespace-definition:
                namespace identifier { namespace-body }

        extension-namespace-definition:
                namespace original-namespace-name  { namespace-body }

        unnamed-namespace-definition:
                namespace { namespace-body }

        namespace-body:
                declaration-seqopt
        namespace-alias:
                identifier

        namespace-alias-definition:
                namespace identifier = qualified-namespace-specifier ;

        qualified-namespace-specifier:
                ::opt nested-name-specifieropt namespace-name
        using-declaration:
                using typenameopt ::opt nested-name-specifier unqualified-id ;
                using ::  unqualified-id ;
        using-directive:
                using  namespace ::opt nested-name-specifieropt namespace-name ;
        asm-definition:
                asm ( string-literal ) ;
        linkage-specification:
                extern string-literal { declaration-seqopt }
                extern string-literal declaration
```

1.7 Declarators **[gram.dcl.decl]**

```
        init-declarator-list:
                init-declarator
                init-declarator-list , init-declarator
        init-declarator:
                declarator initializeropt
        declarator:
                direct-declarator
                ptr-operator declarator
        direct-declarator:
                declarator-id
                direct-declarator ( parameter-declaration-clause ) cv-qualifier-seqopt exception-s
                direct-declarator [ constant-expressionopt ]
                ( declarator )
        ptr-operator:
                * cv-qualifier-seqopt
                &
                ::opt nested-name-specifier * cv-qualifier-seqopt
        cv-qualifier-seq:
                cv-qualifier cv-qualifier-seqopt
        cv-qualifier:
                const
                volatile
        declarator-id:
                ::opt id-expression
                ::opt nested-name-specifieropt type-name
        type-id:
                type-specifier-seq abstract-declaratoropt
        type-specifier-seq:
                type-specifier type-specifier-seqopt
        abstract-declarator:
                ptr-operator abstract-declaratoropt
                direct-abstract-declarator
        direct-abstract-declarator:
                direct-abstract-declaratoropt ( parameter-declaration-clause ) cv-qualifier-seqopt
                direct-abstract-declaratoropt [ constant-expressionopt ]
                ( abstract-declarator )
        parameter-declaration-clause:
                parameter-declaration-listopt ...opt
                parameter-declaration-list , ...
        parameter-declaration-list:
                parameter-declaration
                parameter-declaration-list , parameter-declaration
        parameter-declaration:
                decl-specifier-seq declarator
                decl-specifier-seq declarator = assignment-expression
                decl-specifier-seq abstract-declaratoropt
                decl-specifier-seq abstract-declaratoropt = assignment-expression
        function-definition:
                decl-specifier-seqopt declarator ctor-initializeropt function-body
                decl-specifier-seqopt declarator function-try-block

        function-body:
                compound-statement

        initializer:
                = initializer-clause
                ( expression-list )
        initializer-clause:
                assignment-expression
                { initializer-list ,opt }
                { }
        initializer-list:
                initializer-clause
                initializer-list , initializer-clause
```

1.8 Classes **[gram.class]**

```
        class-name:
                identifier
                template-id
        class-specifier:
                class-head { member-specificationopt }
        class-head:
                class-key identifieropt base-clauseopt
                class-key nested-name-specifier identifier base-clauseopt
        class-key:
                class
                struct
                union
        member-specification:
                member-declaration member-specificationopt
                access-specifier : member-specificationopt
        member-declaration:
                decl-specifier-seqopt member-declarator-listopt ;
                function-definition ;opt
                qualified-id ;
                using-declaration
                template-declaration
        member-declarator-list:
                member-declarator
                member-declarator-list , member-declarator
        member-declarator:
                declarator pure-specifieropt
                declarator constant-initializeropt
                identifieropt : constant-expression
        pure-specifier:
                 = 0
        constant-initializer:
                 = constant-expression
```

1.9 Derived classes **[gram.class.derived]**

```
base-clause:
        : base-specifier-list
base-specifier-list:
        base-specifier
        base-specifier-list , base-specifier

base-specifier:
        ::opt nested-name-specifieropt class-name
        virtual access-specifieropt ::opt nested-name-specifieropt class-name
        access-specifier virtualopt ::opt nested-name-specifieropt class-name
access-specifier:
        private
        protected
        public
```

1.10 Special member functions **[gram.special]**

```
conversion-function-id:
        operator conversion-type-id
conversion-type-id:
        type-specifier-seq conversion-declaratoropt
conversion-declarator:
        ptr-operator conversion-declaratoropt
ctor-initializer:
        : mem-initializer-list
mem-initializer-list:
        mem-initializer
        mem-initializer , mem-initializer-list
mem-initializer:
        mem-initializer-id ( expression-listopt )
mem-initializer-id:
        ::opt nested-name-specifieropt class-name
        identifier
```

1.11 Overloading **[gram.over]**

```
operator-function-id:
        operator operator
operator: one of
        new  delete    new[]     delete[]
        +    -    *    /    %    ^    &    |    ~
        !    =    <    >    +=   -=   *=   /=   %=
        ^=   &=   |=   <<   >>   >>=  <<=  ==   !=
        <=   >=   &&   ||   ++   --   ,    ->*  ->
        ()   []
```

1.12 Templates **[gram.temp]**

```
template-declaration:
        exportopt template < template-parameter-list > declaration
template-parameter-list:
        template-parameter
        template-parameter-list , template-parameter
template-parameter:
        type-parameter
        parameter-declaration
type-parameter:
        class identifieropt
        class identifieropt = type-id
        typename identifieropt
        typename identifieropt = type-id
        template < template-parameter-list > class  identifieropt
        template < template-parameter-list > class  identifieropt = template-name

template-id:
        template-name < template-argument-list >
template-name:
        identifier
template-argument-list:
        template-argument
        template-argument-list , template-argument
template-argument:
        assignment-expression
```

```
                type-id
                template-name
        explicit-instantiation:
                template-declaration
        explicit-specialization:
                template < > declaration

  1.13  Exception handling                                   [gram.except]
        try-block:
                 try compound-statement handler-seq
        function-try-block:
                 try  ctor-initializeropt function-body handler-seq
        handler-seq:
                handler handler-seqopt
        handler:
                catch ( exception-declaration ) compound-statement
        exception-declaration:
                type-specifier-seq declarator
                type-specifier-seq abstract-declarator
                type-specifier-seq
                ...
        throw-expression:
                throw assignment-expressionopt
        exception-specification:
                throw ( type-id-listopt )
        type-id-list:
                type-id
                type-id-list ,  type-id

  1.14  Preprocessing directives                                [gram.cpp]
        preprocessing-file:
                groupopt
        group:
                group-part
                group group-part
        group-part:
                pp-tokensopt new-line
                if-section
                control-line
        if-section:
                if-group elif-groupsopt else-groupopt endif-line
        if-group:
                # if     constant-expression new-line groupopt
                # ifdef  identifier new-line groupopt
                # ifndef identifier new-line groupopt

        elif-groups:
                elif-group
                elif-groups elif-group
        elif-group:
                # elif   constant-expression new-line groupopt
        else-group:
                # else   new-line groupopt
        endif-line:
                # endif  new-line
        control-line:
                # include pp-tokens new-line
                # define  identifier replacement-list new-line
                # define  identifier lparen identifier-listopt ) replacement-list new-line
                # undef   identifier new-line
                # line    pp-tokens new-line
                # error   pp-tokensopt new-line
                # pragma  pp-tokensopt new-line
                #         new-line
        lparen:
                the left-parenthesis character without preceding white-space
        replacement-list:
                pp-tokensopt
        pp-tokens:
                preprocessing-token
                pp-tokens preprocessing-token
        new-line:
                the new-line character
```